2.7 COMM DEVICES

Communication is a cooperative exchange of information between two or more parties. This exchange can be carried out it several ways, including propagation of EM or acoustic energy through the atmosphere (radio, microwave, etc.) or into space, and by propagation of electric or electro-optical impulses along a conductor or optical medium (telephone or fiber optic land line).

Communication consists of a transmission of a signal, propagation of the transmitted signal through some medium (usually the atmosphere, the sea, or space), and reception by a receiver. In order for communication to be successful, several physical factors must be taken into account. The transmitter and receiver must be tuned to nearly the same frequency (bandwidths must overlap). The signal must be transmitted with sufficient power to overcome the effects of any attenuation of the signal by the propagation medium. Finally, the receiver must be sufficiently sensitive to distinguish the signal from background noise and interference from external sources (i.e. jamming or EM coupling of some kind). SWEG implements communication by allowing the user to define EM communication devices and networks. There are two types of communication devices: transmitters and receivers. In the real world, the presence of one device does not imply the presence of the other. However, SWEG requires that transmitters and receivers occur in pairs.

SWEG models both intraplayer and interplayer communications. Intraplayer communication is instantaneous sharing of perceptions. Interplayer communication can be represented either implicitly or explicitly. When explicit modeling of communication is desired, SWEG is capable of simulating many of the physical aspects of communication. The physical aspects of explicit communication are described in this section.

Communications devices are limited in their performance by a variety of effects which can collectively be called internal losses. This noise, plus any induced by jamming, must be overcome in order for an attempt at communication to successful. SWEG provides the user with the ability to define the internal losses and a signal “recognition threshold” for a receiver as well as the power of a transmitter. If the S/I ratio is below this level, the message is not received by the intended recipient.

Transmitters and receivers are limited in the frequency band(s) over which they can transmit/receive. SWEG provides the ability for the user to define the bandwidth for both receivers and transmitters.

Real antennae can be either directional or omni-directional. SWEG allows the definition of an antenna pattern which defines a device's sensitivity (or gain) in a given direction. This pattern is defined in both elevation and azimuth relative to the receiver, and therefore provides a three-dimensional reception/transmission model.

Signals from real world communications devices are often polarized due to their physical construction and to lessen the effects of jamming. SWEG allows the user to define both linear and circular polarization characteristics of receivers, but this is used only for jamming (see Section 2.8).

In the real world, any information transfer is limited by the speed of the medium of signal propagation. In SWEG all signal transmissions are assumed to occur instantaneously.

However, the definition of a communications network allows the specification of a transmission delay. It is possible to include an additional delay to account for signal propagation time in this definition.

2.7.1 Functional Element Design Requirements

This section contains all of the requirements necessary to implement the simulation of communication devices in SWEG.

a. SWEG will provide the capability for a user to define types of communication devices that operate at any frequencies in the EM spectrum. Any type of platform may have any type of communication device.

b. SWEG will model instantaneous signal transmission, but will use a message delay time as defined by the user for each message type on each network type (see Section 2.22).

c. SWEG will provide the capability for the user to place each communication device on either an explicit network or an implicit network. SWEG will model effects of jamming and masking for devices on explicit networks, but not for those on implicit networks.

d. For communication devices on explicit communications networks, SWEG will use the standard one-way energy signal transmission equation to calculate the S/I received. The following user-defined parameters will be included:

   - gain of the receiver type
   - internal receiver type losses
   - receiver frequency
   - receiver type noise
   - transmitter type power
   - transmitter type gain

e. SWEG will provide the capability for the user to define an antenna gain table for each type of communication transmitter and each type of receiver. The table will provide antenna gain values (in dB) for frequency, azimuth, and elevation intervals arbitrarily defined by the user (frequency is optional). The intervals will be discrete and there will be no interpolation of gain between intervals. The user may also define the vertical offset of each antenna from its platform.

f. SWEG will set the frequency of each specific communication device to the frequency of its network at the time of use. (See Section 2.22).

g. SWEG will provide the capability for the user to set the original operational status of each communication device and to point each one at a specific platform or a specific geometric location, or in a specific direction. If the pointing direction is not specified, SWEG will point the device in the same direction as the forward-facing direction of its platform.

h. SWEG will provide the capability for a user to define an emission code for each communications transmitter that will identify it for use in correlating model actions with other simulations in a networked environment. (This requirement has currently led to a violation of the precept that TDB and SDB instructions should contain no instructions specific to either a constructive or a virtual mode of operation).

2.7.2 Functional Element Design Approach

Communication devices are defined in the TDB as system categories embedded in a player-structure (see Section 2.1). The system categories for communication devices are COMM-XMTR and COMM-RCVR. A communication transmitter and a receiver must both be defined if either are desired for a particular type of player.

The definition of the physical and performance characteristics of communications devices is provided in the TDB. A capability is associated with each device. Each capability consists of a set of data items which define the communications device to be modeled.

A net for each specific communications device is chosen in the SDB. The user also defines a frequency and (possibly) alternate frequencies for each network. Alternate frequencies may be defined to allow a player whose communications are being jammed to switch to another frequency.

Communications devices are defined as systems within an element. For example:

```
ELEMENT: 11    cmd_post_ele      DISCRETE QUANTITY: 1
      COMM-RCVR 112 comm_rcvr ON FREQ: 2.3 (GHZ) NET: 11 broadcast
END ELEMENT
```

During a SWEG run, an event is scheduled to initiate an attempt to communicate. The delay time is added to the current game time to determine the anticipated time of receipt. If communications are being modeled implicitly, no signal calculations are done and the transmission is assumed to be successful.

If communications are being modeled explicitly, the signal level is calculated at the receiver. This is done by checking to see if a signal level has already been calculated for the current transmitter/receiver pair or if they are in relative motion, which would change the signal level. Once the positions have been updated, the relative antenna orientations are determined and the range and gain are calculated. If the signal level is greater than zero, jamming interactions (if modeled) are calculated.

Next, the signal level is compared to the threshold for the receiver. If the threshold is exceeded, the message is received, and an event is scheduled for the recipient to notice it. If the message is not received, the transmitter will try an alternate frequency, if available.

2.7.3 Functional Element Software Design

This section contains a table and two software code trees which describe the software design necessary to implement the requirements and design approach outlined above. Table 2.7-1 lists most of the functions found in the code trees, and a description of each function is provided. Figure 2.7-1 depicts the path from main to yakker, the top-level C++ function within the code for communication events. Figure 2.7-2 contains the code tree for yakker and its subordinate functions.

A function's subtree is provided within the figure only the first time that the function is called. Some functions are extensively called throughout SWEG, and the trees for these functions are in the appendix to this document rather than within each FE description. Within this FE, the functions in that category are MITRcontrol and all member functions in the C++ class WhereIsIt.

Not all functions shown in the figures are included in the table. The omitted entries are trivial lookup functions (single assignment statements), list-processing or memory allocation functions, or C++ class functions for construction, etc.

TABLE 2.7-1. Communications Functions.

Function	**Description**
antgeom	calculates antenna pointing and relative angle data
BaseHost::Run	runs all steps
crslwc	determines line/circle crossing
crslwl	determines line/line crossing
crslwp	determines line/plane crossing
ergatn	calculates attenuation for energy transmission
ergazel	retrieves table entry for gain from azimuth and elevation
erggar	calculates gain and range for energy transmission
featlos	determines if shapes interfere with line of sight
jamcal	calculates jammer interference power at victim receiver
loschk	checks line of site between two objects
main	controls overall execution
MainInit	initiates processing
MainParse	controls parsing of user instructions
numerical	sorts function for address codes
program	controls execution of all steps except bootstrap
redwood	adds new entry to or removes top entry from leftist tree
region	determines if a point is within a two dimensional region
semant	controls semantic processing of instructions
simnxt	controls runtime execution
simphy	controls processing physical events
simul8	controls semantic processing of runtime instructions
srhpro	searches table for an entry containing a specific value
TAddrData::GetJamInteractions	checks for jammer interactions with a communications device
TAddrData::GetParentData	retrieves the TAddrData object from a parent
TAddrData::GetShapeList	finds the shape list at a given address
TAddress::GetAddresses	retrieves a sorted collection of addresses between two points
TAddress::InsertVertCodes	inserts address codes, including parents, for the given point
TMemory::Allocate	allocates permanent storage
TMemory::AllocTemp	allocates temporary storage

TABLE 2.7-1. Communications Functions. (Contd.)

Function	Description
TMemory::Deallocate	deallocates storage
TMemory::DeallocFront	deallocates storage
TMemory::LLSTremove	removes a node from a linked list
TMemory::LLSTsearch	searches a list
TMemory::LLSTsearchhard	searches a list using extra parameters
TTerrain::EdgeMasklos	determines the masking of terrain edges
TTerrain::Elevation	determines the z-coordinate on a surface given the x and y coordinates
TTerrain::FindTriangle	determines the terrain triangle for a point given an x, y coordinate pair
TTerrain::LineOfSight	determines if there is a line of sight between two objects
WhereIsIt::CalcPosition	determines position for a platform given a time
WhereIsIt::CalcUnitVel	determines unit velocity for a platform given a time
WhereIsIt::CalcUpVector	determines local up vector given a time
yaeail	adds yet another entry to the scenario action item list
yakker	controls processing of communications events
yaknex	determines next communications event for a given net
yaksig	determines signal level at receiver

```
main
    |-BaseHost::Run
        |-MainInit
            |-program
                |-MainParse
                    |-semant
                        |-simul8
                            |-simnxt
                                |-simphy
                                    |-yakker
                                        |-yaksig
                                        |-yaknex
```

FIGURE 2.7-1. Top-Level Communications Code Tree.

```
yakker
    |-TMemory::Index2Ptr
    |-TTable::SearchIGetDouble
    |   \-TTable::RequiredInt
    |       |-TTable::SearchInteger
    |       |   \-TMemory::Ptr2Index
    |       \-TMessages::WriteMessage
    |           |-TMessages::GetMsg
    |           |-TMessages::PrintALine
    |           |   \-TSeqFile::Write
    |           |       \-MFiles::Append
    |           \-sysdun
    |               |-TActWindow::GetNext
    |               |-TActWindow::~TActWindow
    |               \-ProgramStop::ProgramStop
    |                   \-SwegExcpt::SwegExcpt
    |                       \-StringDup
    |-TMaster::GetPlayer
    |   \-TAccDirect::GetItem
    |       \-TAccDirect::GetItem
    |-TPlayer::IsAlive
    |-TMemory::LLSTsearch
    |   |-TMemory::LLSTsearch
    |   |   \-TMemory::Index2Ptr
    |   \-TMemory::Index2Ptr
    |-TMemory::Ptr2Index
    |-yaksig
    |   |-TMemory::Ptr2Index
    |   |-TMemory::LLSTsearch
    |   |-TMemory::LLSTsearchhard
    |   |   |-TMemory::LLSTsearchhard
    |   |   |   \-TMemory::Index2Ptr
    |   |   \-TMemory::Index2Ptr
    |   |-TMemory::Allocate
    |   |   \-TMemory::DoAllocate
    |   |       |-TMemory::GetBlockLength
    |   |       |-CountMemOpns
    |   |       |   \-TMaster::DebugOn
    |   |       |-TMemory::Ptr2Index
    |   |       |-ProgramStop::ProgramStop
    |   |       \-TMemory::WriteSummary
    |   |           |-TMemory::WordsUsed
    |   |           |-TMemory::CalcWdsLeft
    |   |           \-CountMemOpns
    |   |-isZeroEquiv
    |   |-TMemory::Index2Ptr
    |   |-WhereIsIt::CalcPosition
    |   |-loschk
```

FIGURE 2.7-2. Communications Code Tree.

```
|    |    |-TMaster::GetTerrain
|    |    |-TMaster::TerrainOn
|    |    |-DVector::Getz
|    |    |-dbg_acos
|    |    |    |-MathExcpt::MathExcpt
|    |    |    \-acos_c
|    |    |-operator-
|    |    |-DVector::GetHorizLength
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    \-TTerrain::LineOfSight
|    |         |-DVector::Getz
|    |         |-DVector::Getx
|    |         |-DVector::Gety
|    |         |-dist
|    |         |-dbg_acos
|    |         |-VertexIndex::VertexIndex
|    |         |    \-VertexIndex::operator=
|    |         |-TTerrain::FindTriangle
|    |         |    |-VertexIndex::VertexIndex
|    |         |    |-TAddress::Cartesian2Spherical
|    |         |    |    |-dbg_sqrt
|    |         |    |    |-dbg_asin
|    |         |    |    |    |-MathExcpt::MathExcpt
|    |         |    |    |    \-asin_c
|    |         |    |    \-dbg_atan2
|    |         |    |         \-MathExcpt::MathExcpt
|    |         |    |              \-SwegExcpt::SwegExcpt
|    |         |    |-TMessages::WriteMessage
|    |         |    |-TAddress::Spherical2Cartesian
|    |         |    |    \-TMaster::DebugOn
|    |         |    |-VertexIndex::operator=
|    |         |    |-VerticeArray::operator[]
|    |         |    |    \-VertexIndex::Value
|    |         |    |-TTerrain::toPtr
|    |         |    |    \-SwegExcpt::SwegExcpt
|    |         |    |-operator-
|    |         |    |    \-VertexIndex::VertexIndex
|    |         |    |-VertexIndex::operator++
|    |         |    |-dist
|    |         |    |-operator+
|    |         |    |    \-VertexIndex::VertexIndex
|    |         |    \-VertexIndex::operator+=
|    |         |         \-operator+
|    |         \-TTerrain::EdgeMasklos
|    |              |-dist
|    |              |-VerticeArray::operator[]
|    |              |-operator+
```

FIGURE 2.7-2. Communications Code Tree. (Contd.)

```
|   |           |-VertexIndex::operator+=
|   |           |-isZeroEquiv
|   |           \-TTerrain::toIndex
|   |               \-SwegExcpt::SwegExcpt
|   |-featlos
|   |   |-TMaster::GetTerrain
|   |   |-TMaster::TerrainOn
|   |   |-DVector::Getx
|   |   |-DVector::Gety
|   |   |-DVector::Getz
|   |   |-TTerrain::Elevation
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   |-VertexIndex::VertexIndex
|   |   |   |-TTerrain::FindTriangle
|   |   |   |-VerticeArray::operator[]
|   |   |   |-operator+
|   |   |   \-isZeroEquiv
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::AllocTemp
|   |   |   \-TMemory::DoAllocate
|   |   |-sorted_collection::sorted_collection
|   |   |-TAddress::GetAddresses
|   |   |   |-TAddress::GetCode
|   |   |   |   |-DVector::Getx
|   |   |   |   |-DVector::Gety
|   |   |   |   \-TMessages::WriteMessage
|   |   |   |-TAddress::InsertVertCodes
|   |   |   |   |-sorted_collection::insert_nodup
|   |   |   |   |   |-MTree::insert_nodup
|   |   |   |   |   |   |-MTree::insert_nodup
|   |   |   |   |   |   \-MTree::MTree
|   |   |   |   |   \-MTree::MTree
|   |   |   |   \-numerical
|   |   |   |-operator-
|   |   |   |-operator^
|   |   |   |-TAddress::GetCellRadius
|   |   |   |-dbg_sqrt
|   |   |   |-operator*
|   |   |   |-DVector::operator+=
|   |   |   \-operator+
|   |   |-sorted_collection::getfirst
|   |   |-sorted_collection::getnext
|   |   |   \-MTree::getnext
|   |   |-TAddress::GetShapeList
|   |   |   |-TAddrNode::DataPresent
|   |   |   |   \-TAddrNode::GetNode
|   |   |   |       |-TAddrData::GetCode
```

FIGURE 2.7-2. Communications Code Tree. (Contd.)

```
|   |   |   |       |-TAddrData::TAddrData
|   |   |   |       |-TAddrNode::TAddrNode
|   |   |   |       |   \-MTree::MTree
|   |   |   |       \-TAddrData::PutNodePtr
|   |   |   \-TAddrData::GetShapeList
|   |   |       \-TMemory::Index2Ptr
|   |   |-TMemory::Ptr2Index
|   |   |-TTable::SearchInt
|   |   |   \-TMemory::Ptr2Index
|   |   |-region
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   |-dbg_atan2
|   |   |   \-dist
|   |   |-crslwp
|   |   |   |-DVector::DVector
|   |   |   |-DVector::DVector
|   |   |   |-dbg_sqrt
|   |   |   |-crslwc
|   |   |   |   |-operator-
|   |   |   |   |-DVector::Putz
|   |   |   |   |-operator^
|   |   |   |   |-dbg_sqrt
|   |   |   |   |-operator+
|   |   |   |   \-operator*
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-DVector::operator
|   |   |   |-crslwl
|   |   |   |   |-operator-
|   |   |   |   |-operator*
|   |   |   |   |-DVector::Getz
|   |   |   |   |-operator+
|   |   |   |   \-operator*
|   |   |   |-TMemory::Allocate
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   \-TMemory::Ptr2Index
|   |   |-DVector::DVector
|   |   |-sorted_collection::delete_collection
|   |   |   \-MTree::delete_tree
|   |   |       |-MTree::delete_tree
|   |   |       \-TMemory::Deallocate
|   |   |           |-TMemory::Deallocate
|   |   |           |   |-TMemory::DeallocFront
|   |   |           |   |   \-TMemory::GetBlockLength
|   |   |           |   |-TMemory::Index2Ptr
|   |   |           |   |-CountMemOpns
|   |   |           |   \-TMemory::RcylBlock
```

FIGURE 2.7-2. Communications Code Tree. (Contd.)

```
|    |    |              |         |-TMemory::Index2Ptr
|    |    |              |         \-TMemory::Ptr2Index
|    |    |              \-TMemory::Ptr2Index
|    |    \-TMemory::Deallocate
|    |-erggar
|    |    |-antgeom
|    |    |    |-DVector::DVector
|    |    |    |-WhereIsIt::CalcPosition
|    |    |    |-DVector::operator=
|    |    |    |    |-FloatVector::Getx
|    |    |    |    |-FloatVector::Gety
|    |    |    |    \-FloatVector::Getz
|    |    |    |-DVector::DVector
|    |    |    |    |-FloatVector::Getx
|    |    |    |    |-FloatVector::Gety
|    |    |    |    \-FloatVector::Getz
|    |    |    |-operator-
|    |    |    |-DVector::Norm
|    |    |    |-DVector::GetHorizLength
|    |    |    |-DVector::operator
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Getz
|    |    |    |-DVector::Gety
|    |    |    |-WhereIsIt::CalcUnitVel
|    |    |    |-WhereIsIt::CalcUpVector
|    |    |    |-operator*
|    |    |    |-operator+
|    |    |    |-operator*
|    |    |    |-FloatVector::Getx
|    |    |    |-FloatVector::Gety
|    |    |    |-FloatVector::Getz
|    |    |    |-DVector::VecAng
|    |    |    |    |-operator^
|    |    |    |    |-operator*
|    |    |    |    |-DVector::DVector
|    |    |    |    |-DVector::GetLength
|    |    |    |    |-isZeroEquiv
|    |    |    |    \-dbg_atan2
|    |    |    \-DVector::GetLength
|    |    |-TMemory::Index2Ptr
|    |    |-srhpro
|    |    |-TMemory::Ptr2Index
|    |    \-ergazel
|    |         |-TMemory::Index2Ptr
|    |         |-srhpro
|    |         \-TMemory::Ptr2Index
|    |-DVector::Getz
|    |-operator-
```

FIGURE 2.7-2. Communications Code Tree. (Contd.)

```
|    |-DVector::GetHorizLength
|    |-ergatn
|    |    \-srhpro
|    \-dbg_pow
|         \-MathExcpt::MathExcpt
|-jamcal
|    |-TMemory::Index2Ptr
|    |-WhereIsIt::CalcPosition
|    |-TAddrData::GetParentData
|    |    \-TAddrNode::GetParentData
|    |-TAddrData::GetJamInteractions
|    |-TTable::SearchInt
|    \-TMemory::Deallocate
|-TMemory::Allocate
|-yaeail
|    |-TMemory::Index2Ptr
|    |-TMaster::GetGameTime
|    |-TMaster::DebugOn
|    |-TMessages::WriteMessage
|    |-TMemory::Allocate
|    |-TMemory::Ptr2Index
|    |-TMaster::PutScenrTree
|    |-redwood
|    |    |-TMemory::Index2Ptr
|    |    \-TMemory::Ptr2Index
|    |-TMaster::GetScenrTree
|    |    \-TMemory::Index2Ptr
|    \-TMaster::GetPhase
|-yaknex
|    |-yaeail
|    |-TMemory::LLSTremove
|    |    |-TMemory::Index2Ptr
|    |    |-TMemory::LLSTsearch
|    |    |    \-TMemory::Index2Ptr
|    |    \-TMemory::Deallocate
|    |         |-TMemory::Ptr2Index
|    |         |-TMemory::DeallocFront
|    |         |-CountMemOpns
|    |         \-TMemory::RcylBlock
|    \-TMemory::Deallocate
\-TMemory::Deallocate
```

FIGURE 2.7-2. Communications Code Tree. (Contd.)

2.7.4 Assumptions and Limitations

- Energy transmission is instantaneous.
- Energy use is not explicitly represented.
- The speed of light is 299,792,800 m/sec and the speed of sound is 330.28 m/sec.
- Signal polarization does not affect message reception.

- Root mean square signal power is used in all calculations.
- Received power is uniformly distributed over the receiver bandwidth.
- Communication transmitters and receivers must always be defined in pairs.
- Communications signals in SWEG are assumed to propagate instantaneously.

2.7.5 Known Problems or Anomalies

None.